Using the iKart mobile platform

Marco Randazzo

Sestri Levante, July 23th, 2012

iKart

General description

A holonomic mobile base for iCub:

- Omnidirectional movement (six omnidirectional wheels)
- Integrates an high-performance i7 quad core CPU
- Wireless connection (300Mbs)
- Lithium battery 48V 20Ah (approx.2 hours of autonomy)
- Laser scanner (Hokuyo utm-30lx) for obstacle detection

Control software available on iCub SVN repository (opensource):

- Joystick control module (*main/tools/joystickCtrl*)
- Laser scanner (*yarp device driver + ikart/laserScannerGui*)
- Battery management (*iKartbatteryManager + iKartBatteryDisplay)*
- iKart control (*iKartCtrl)*
- Simple navigation module (*iKartGoto*)
- Reactive navigation module (*iKartNav*)
- Force control module *(ikart/forceGuidance)*

Manual:

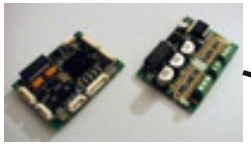
- http://eris.liralab.it/wiki/IKart

iKart

iKart Hardware

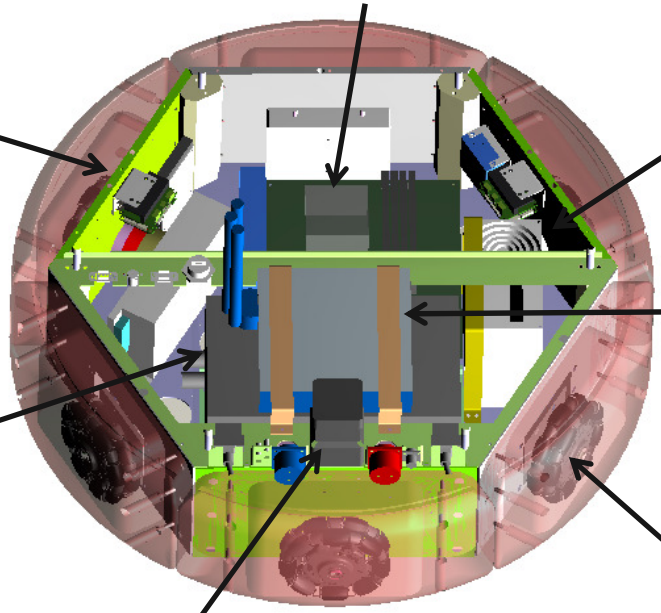
Main H/W components:

Intel i7 motherboard



Motor control boards

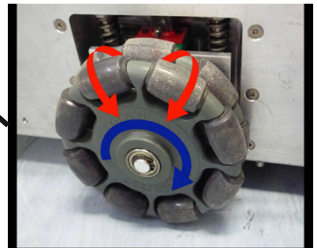
ATX Power supply &
DC/DC converter



Wireless access point

Lithium battery 48V 20Ah

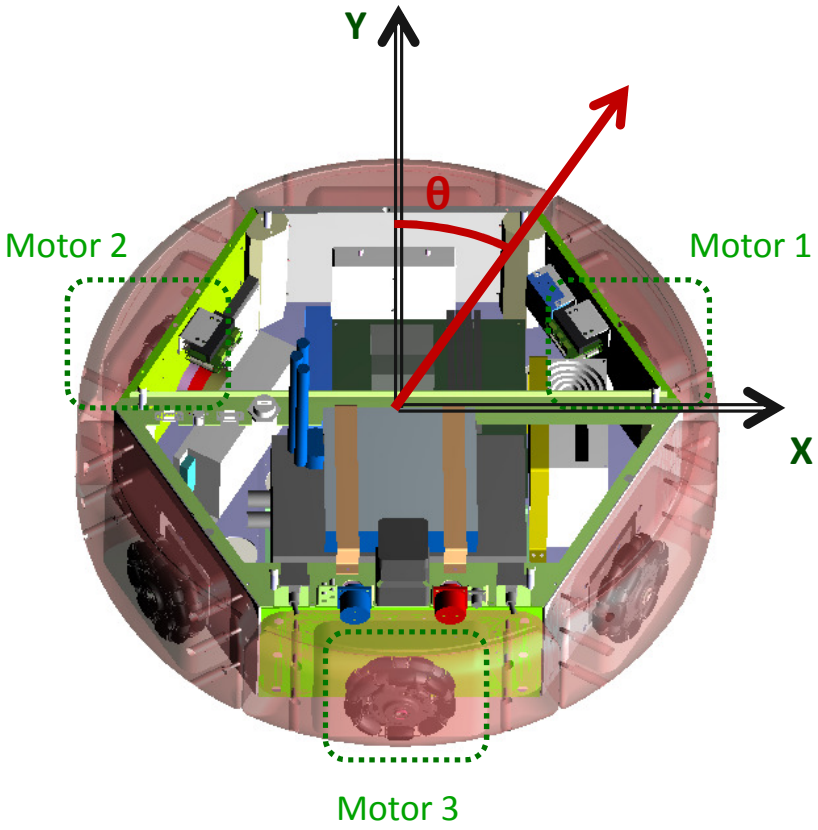
Hokuyo laser scanner



omnidirectional wheels

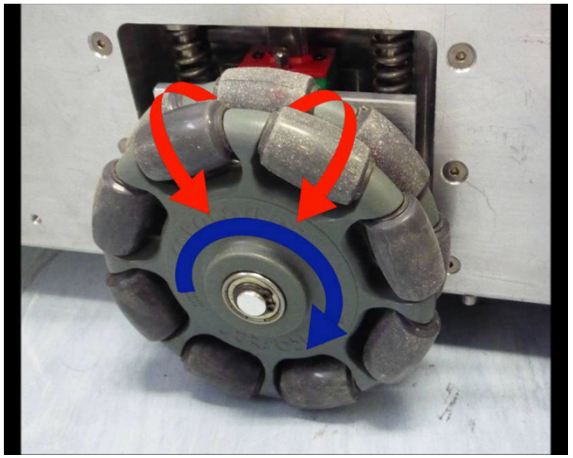
iKart

iKart Hardware



- Three actuated wheels (kollmorgen 48V brushless motors, 100:1 harmonic reduction)
- Three idle wheels (with tunable suspensions) to improve platform stability.

iKart entering in an office



“Rotacaster” omnidirectional wheel

iKart

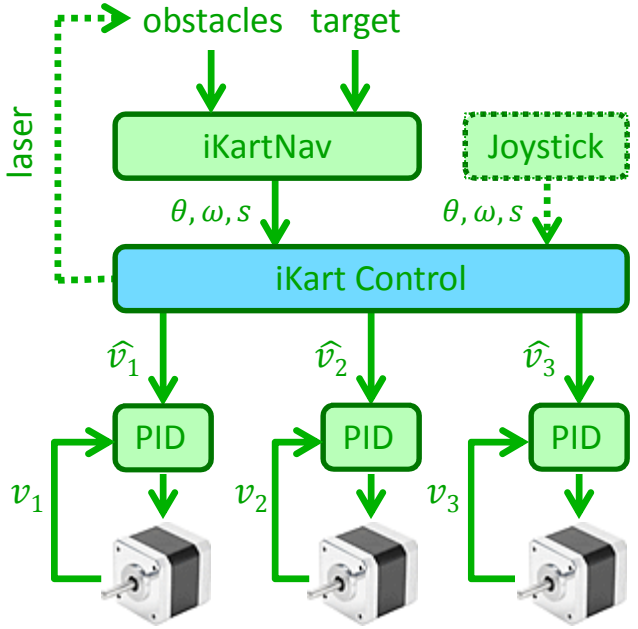
iKart Control Module (iKartCtrl)

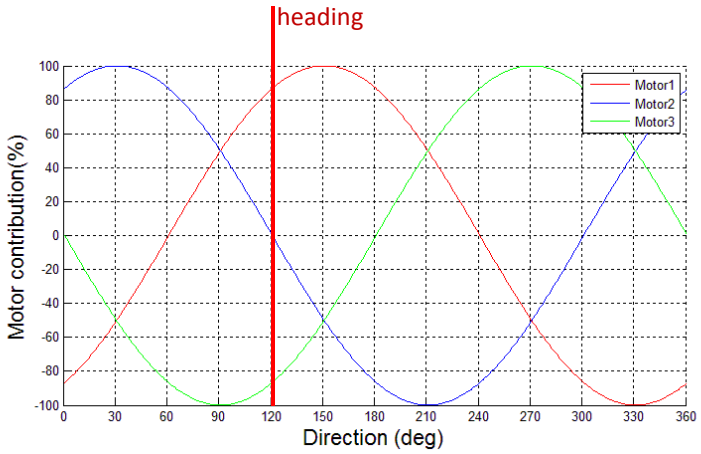
iKart Control Module:

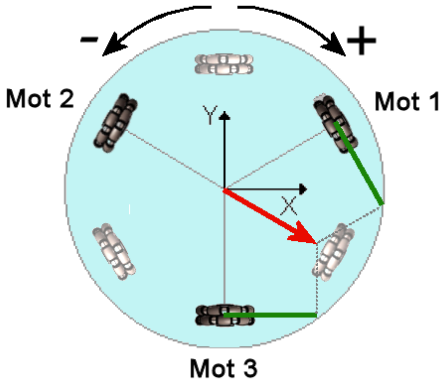
- Two input ports: a general command port (user application) and a priority port (joystick control).

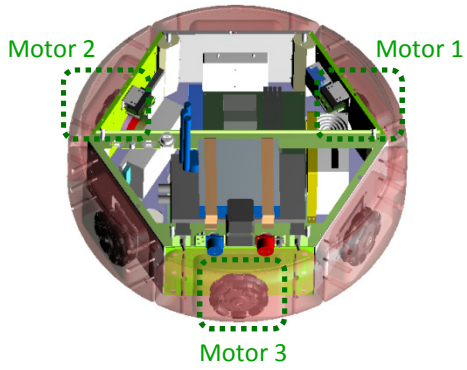
- Computes the speed references for the low level PID motor controllers. The motors commands are obtained from the Cartesian commands according to the following formula:

$$\text{Mot1} = speed_{linear} * \cos(150 - heading) + speed_{angular}$$
$$\text{Mot2} = speed_{linear} * \cos(30 - heading) + speed_{angular}$$
$$\text{Mot3} = speed_{linear} * \cos(270 - heading) + speed_{angular}$$









iKart Basics

- Basic controls
- Odometry vs map localization
- Basic navigation modules

iKart

Basic controls

Starting/stopping the iKart

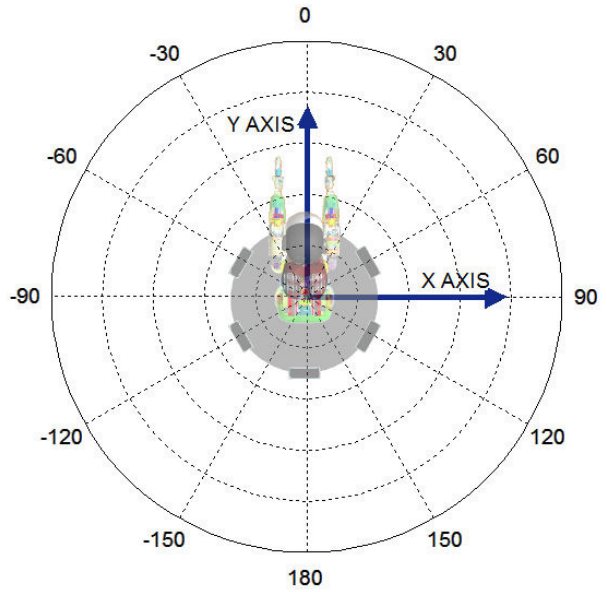
- To start the iKart Launch the script: ikart_start.sh
- To stop the iKart Launch the script ikart_stop.sh

More info on the iKart manual: http://eris.liralab.it/wiki/IKart

Controlling the iKart

- To read the odometry data:
  yarp read … /ikart/odometry:o

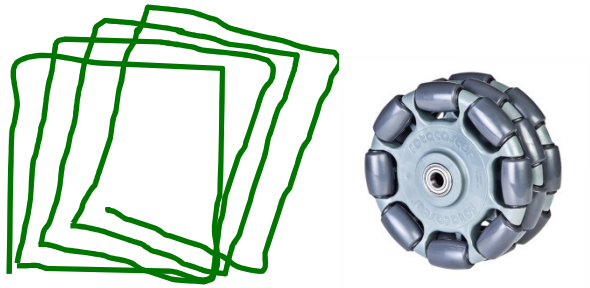
- Velocity commands expressed in the iKart reference frame have to be sent to the port /ikart/control:I or /ikart/aux_control:I
- Commands can be expressed both in Cartesian or polar coordinates: http://eris.liralab.it/wiki/IKart



Localization methods:

ODOMETRY BASED:

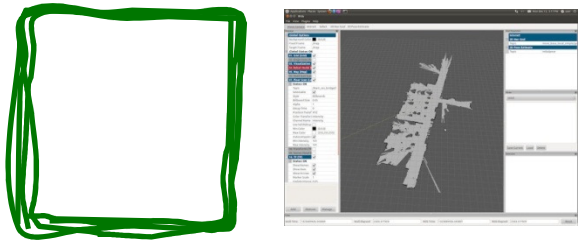
- Simple: directly computed by iKartCtrl using the wheels encoders
- The root reference frame is fixed at the startup (can be reset by the user)
- Affected by integral error



/ikart/odometry:o

MAP/LASER BASED:

- Accurate and repeteable
- The root reference frame is fixed in the map
- Computed by external user modules (e.g.: ROS amcl) using both odometry and laser data



/ikart_ros_bridge/localization:o

iKart

Localization with map

Building a map

(using ROS gmapping, more on ROS later)

How to launch it:

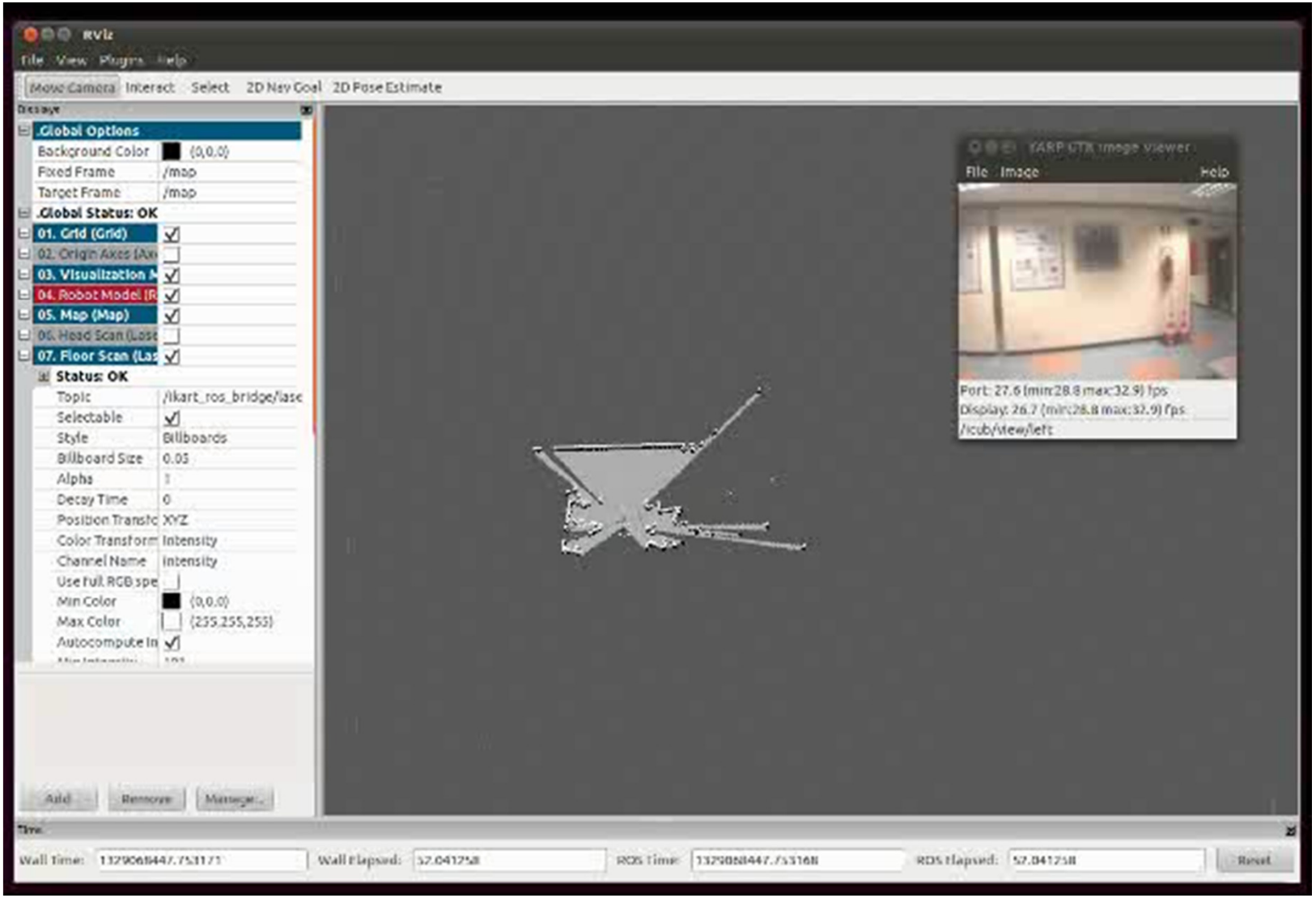
- Verify on the ROS machine that roscore is up and running.
- Verify on the ROS machine that the YARP server running on the iKart can be accessed (yarp detect)
- Start the iKartRosBridge ($IKART_ROS_BRIDGE/bin/iKartRosBridge)
- Start the mapping module ($IKART_ROS_BRIDGE/launch/ikart_build_map.launch)

How to use it:

- Move around the iKart at SLOW speed.
- Save the current map with the script: ($IKART_ROS_BRIDGE/launch/save_map.sh <map_name>)
- Stop the mapping module AFTER saving the map.
- You can reload the map with the script: ($IKART_ROS_BRIDGE/launch/load_map.sh <map_name>)

iKart

Building a map



iKart

Navigation

Available navigation modules:

iKartGoto:

- iKart simply moves towards the user specified goal in a straight line. If an obstacle is encountered the navigation stops.
- iKartRosBridge required if laser-map localization is used.

- Laser-map based
- Odometry based

iKartNav:

- Reactive navigation system based on Artificial Potential Fields.
- Goal should be provided in a closed-loop form.

- Odometry based

ROS Navigation stack:

- Complex navigation tasks (i.e. go from a room to another, avoiding people etc.)
- The navigation stack parameters have to be tuned for the iKart.
- iKartRosBridge required.

- Laser-map based

iKart

Navigation

iKartGoto: reach the goal using a basic straight trajectory

How to launch it:

(If odometry is used instead of map based localization, skip the ROS steps in blue)

- Verify on the ROS machine that roscore is up and running.
- Verify on the ROS machine that the YARP server running on the iKart can be accessed (yarp detect)
- Start the iKartRosBridge ($IKART_ROS_BRIDGE/bin/iKartRosBridge)
- Start the map server and load the map ($IKART_ROS_BRIDGE/launch/load_maps.sh <map name>)
- Start the localization module (roslaunch $IKART_ROS_BRIDGE/launch/ikart_localize.launch)
- Verify that the localization is correct (roslaunch $IKART_ROS_BRIDGE/lauch/rviz_navigate.launch)
- Start the iKartGoto module

How to use it:

- Connect to the port /ikart/goto/rpc:i
- Available commands:
  - goto <x> <y> <angle>
  - stop
- Joystick always has the priority!

iKart

Navigation

iKartNav: Artificial Potential Fields reactive navigation

How to launch it:

- From the gYarpManager script: iKartNav.xml

How to use it:

- Connect to the port /ikartNav/user:i
- Available commands:
  - target <x> <y> <angle>
  - stop/pause/go
- Joystick always has the priority!

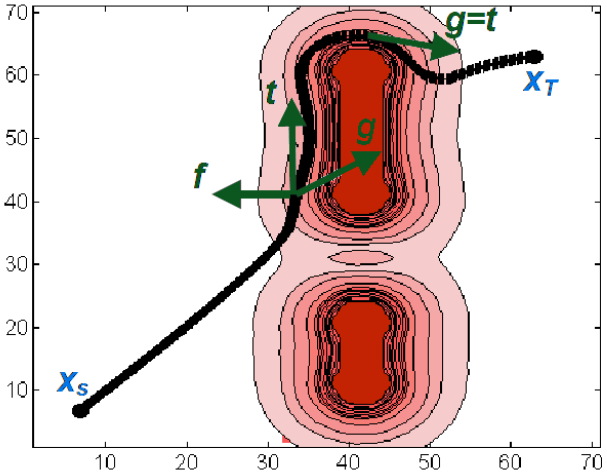
iKart

Reactive Navigation Module (iKartNav)

The navigation function:

$$v = [W_g(U) \cdot \vec{g} + W_t(U) \cdot \vec{t} + W_f(\mathrm{U}) \cdot \vec{f}] \cdot v_{ref}$$

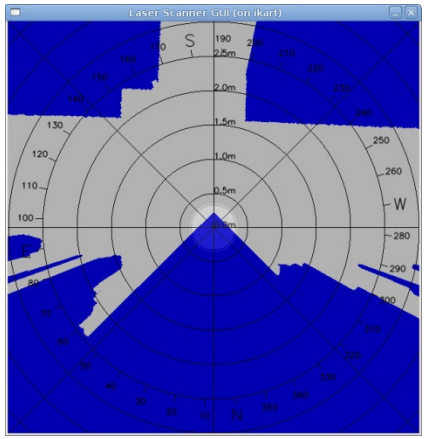
- $W_g = 1 - U_M$ -> goal attractor
- $W_t = U_M$ -> follows obstacles contour
- $W_f = \frac{P - O_i}{|P - O_i|}(U_M)^2$ -> repulsion from obstacles



Simulated reactive navigation

The artificial potential field:

- $U_M = max\{U_i\} \in |0,1|$
- $U_i = \begin{cases} e^{-\frac{1}{2}\cdot\frac{(P-O_i)^2 - R^2}{R^2}} & |P - O_i| \geq R \\ 1 & |P - O_i| < R \end{cases}$



The iKart laser scanner GUI

Navigation using ROS

- iKart–ROS bridge
- Using the navigation stack

iKart

Yarp and ROS

- iKart

Omnidirectional mobile platform for iCub. Battery powered, wireless connected, equipped with laser range finder etc etc…

Yarp Device Drivers
(motor control, laser etc)

- SLAM

Address the problem of simultaneously localizing and building the environment map. An unbiased map is needed for localization while an accurate pose estimate is needed to build the map.

- Navigation

Reach a given point on the map, avoiding obstacles.

ROS modules

iKart

The 'YARP' World

iKartBatteryManager

/ikart/battery:o

iKartBatteryDisplay

/batteryMonitor:i

User module
(e.g. navigation)

JoystickCtrl

/joystickCtrl:o

/ikart/command:i

/ikart/joystick:i

/ikart/odometry:o

/ikart/odometer:o

iKartCtrl

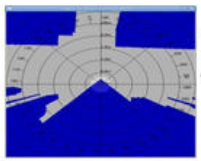
/ikart/laser:o

Device driver

Device driver

iCubInterface

Laserscanner GUI



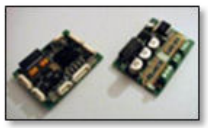
/laserScannerGui/laser:i

Hokuyo
Laser range finder

$\hat{v}_1$ $\hat{v}_2$ $\hat{v}_3$

PID PID PID

v_1 v_2 v_3



Motor control Boards
(firmware level)

iKart

The 'ROS' World

Key modules (ROS Nodes)

- gMapping (SLAM): performs an efficient Rao-Blackwellized particle filter to learn grid maps from laser range data.
  - It is a ROS wrapper aorund an open source library: http://openslam.org/gmapping.html
  - ORIGINAL PAPER: Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard*: Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters*, IEEE Transactions on Robotics, 2006
- map_server: read/writes maps on disk.
- rviz: the ROS gui that display the robot position it he map and allows to send navigation goals.
- amcl: a probabilistic localization system for a robot moving in 2D. It implements an adaptive (KLD-sampling) Monte Carlo localization technique, using a particle filter to track the robot pose against a known map.
  - Derived from a previous Player module. The original algorithm was developed by D. Fox (Intel Research Lab)
  - ORIGINAL PAPERS: D. Fox, W. Burgard, F. Dellaert, and S. Thrun: Monte carlo localization: Efficient position estimation for mobile robots. *AAAI-99*; D. Fox, W. Burgard, F. Dellaert, and S. Thrun Particle filters for mobile robot localization**.** *Sequential Monte Carlo Methods in Practice*. Springer Verlag, New York, 2000.
- move_base: performs navigation tasks in a known map.
  - Developed @ Willow garage. PAPER: Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B.P., and Konolige, K.: *The Office Marathon: Robust Navigation in an Indoor Office Environment*, ICRA 2010.
  - It consists of a global planner (which creates a set of waypoints through the map) and a local planner (which moves the robot to next waypoint, avoiding the obstacles)

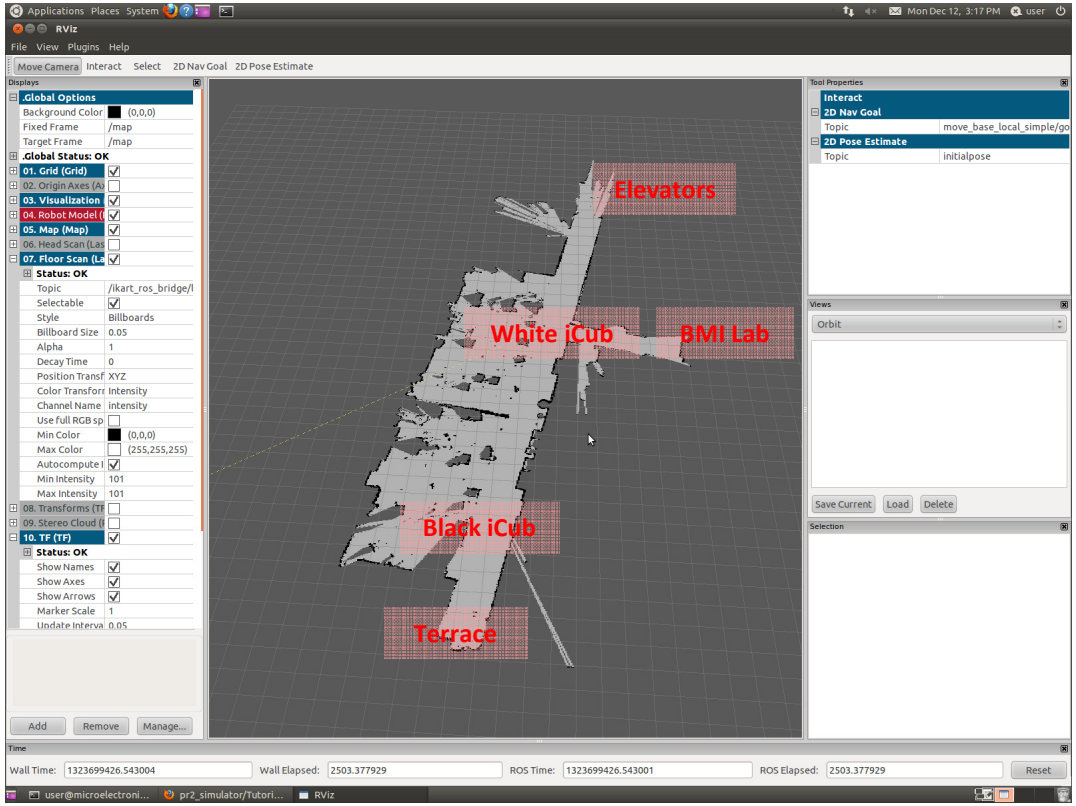
iKart

The 'ROS' World

gMapping node: http://www.ros.org/wiki/gmapping

The map is represented as a standard bitmap:

- Each pixel corresponds to 20cm (scale can be changed).
- White pixels represent free space.
- Black pixel represent walls.
- Once the mapping process is completed, the map cannot be updated anymore. It can be edited, however with a standard picture editor, in order to clean it from noise, add additional walls , etc.
- During the mapping process, the module attempts to transform each incoming laser scan into the odometry frame. The tf node keeps track of the relationship between the different reference frames, broadcasting the */tf* topic.

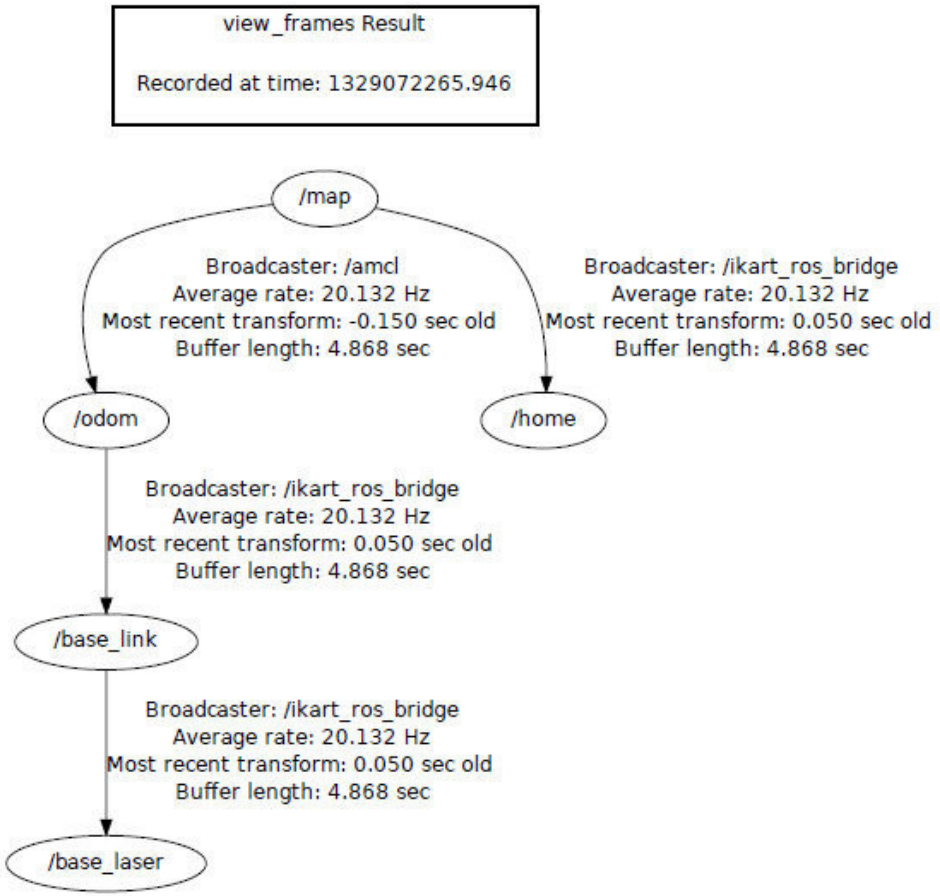


iKart

The 'ROS' World

The Tf package: http://www.ros.org/wiki/tf

tf is a package that maintains the relationship between coordinate frames in a tree structure buffered in time, and lets the user transform points, vectors, etc between any two coordinate frames at any desired point in time.



rostopic echo /tf (= yarp read)

```
transforms:
 -
  header:
   seq: 0
   stamp:
    secs: 1329072390
    nsecs: 487353884
   frame_id: /base_link
  child_frame_id: /base_laser
  transform:
   translation:
    x: 0.245
    y: 0.0
    z: 0.2
   rotation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 1.0
```

```
transforms:
 -
  header:
   seq: 0
   stamp:
    secs: 1329072390
    nsecs: 487353884
   frame_id: /map
  child_frame_id: /home
  transform:
   translation:
    x: 0.0
    y: 0.0
    z: 0.0
   rotation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 1.0
```

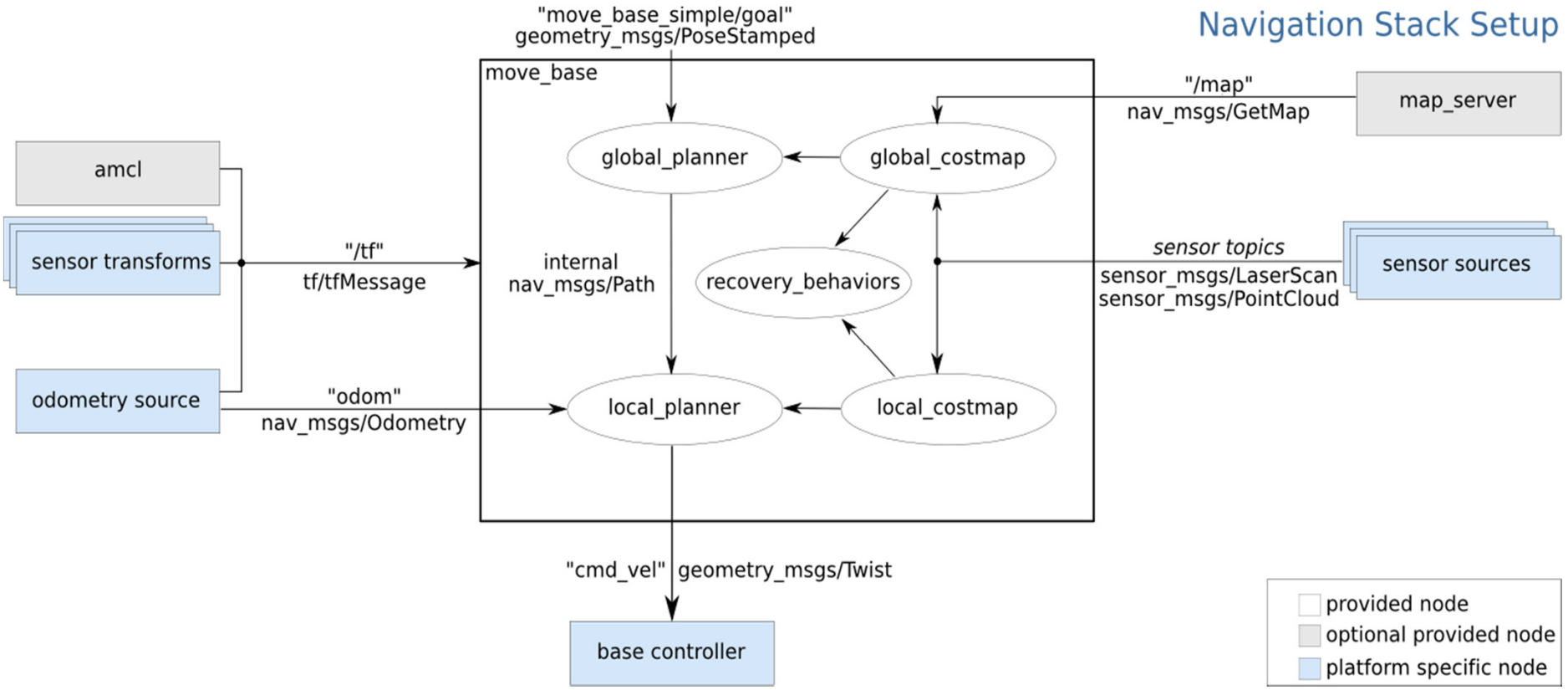
```
transforms:
 -
  header:
   seq: 0
   stamp:
    secs: 1329072390
    nsecs: 487353884
   frame_id: /odom
  child_frame_id: /base_link
  transform:
   translation:
    x: -0.514507994052
    y: -2.9984712994
    z: 0.0
   rotation:
    x: 0.0
    y: 0.0
    z: 0.802155463376
    w: 0.59711524229
```

```
transforms:
 -
  header:
   seq: 0
   stamp:
    secs: 1329072390
    nsecs: 687353884
   frame_id: /map
  child_frame_id: /odom
  transform:
   translation:
    x: -1.89859107801
    y: -1.23302616632
    z: 0.0
   rotation:
    x: 0.0
    y: 0.0
    z: 0.996351233349
    w: -0.085347640878
```

iKart

The 'ROS' World

Move base node: http://www.ros.org/wiki/move_base

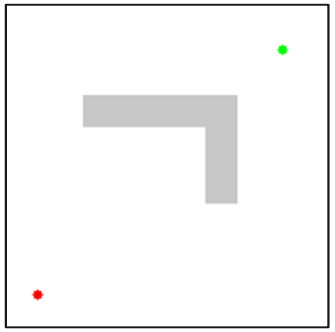


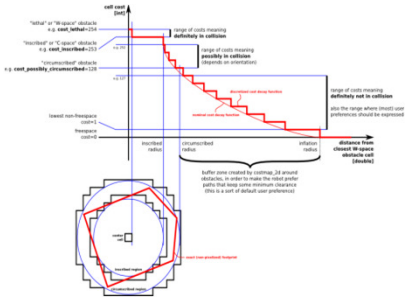
iKart

The 'ROS' World

Global planner: http://www.ros.org/wiki/navigation/GlobalPlanner
http://www.ros.org/wiki/costmap_2d
http://ros.org/wiki/navfn

Given a goal, the global planner creates a series of waypoints for the local planner to achieve. The planner assumes a circular robot and operates on a costmap to find a minimum cost plan from a start point to an end point in a grid. The navigation function is computed with Dijkstra's algorithm.



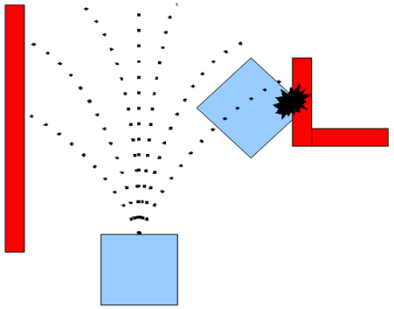


Base local planner: http://ros.org/wiki/base_local_planner

The base_local_planner computes the speed command required to move from the current position to the next waypoint. It uses a 2D grid costmap, different from the one used by the global planner.

Uses Trajectory Rollout / Dynamic Window Approach (DWA) algorithms:

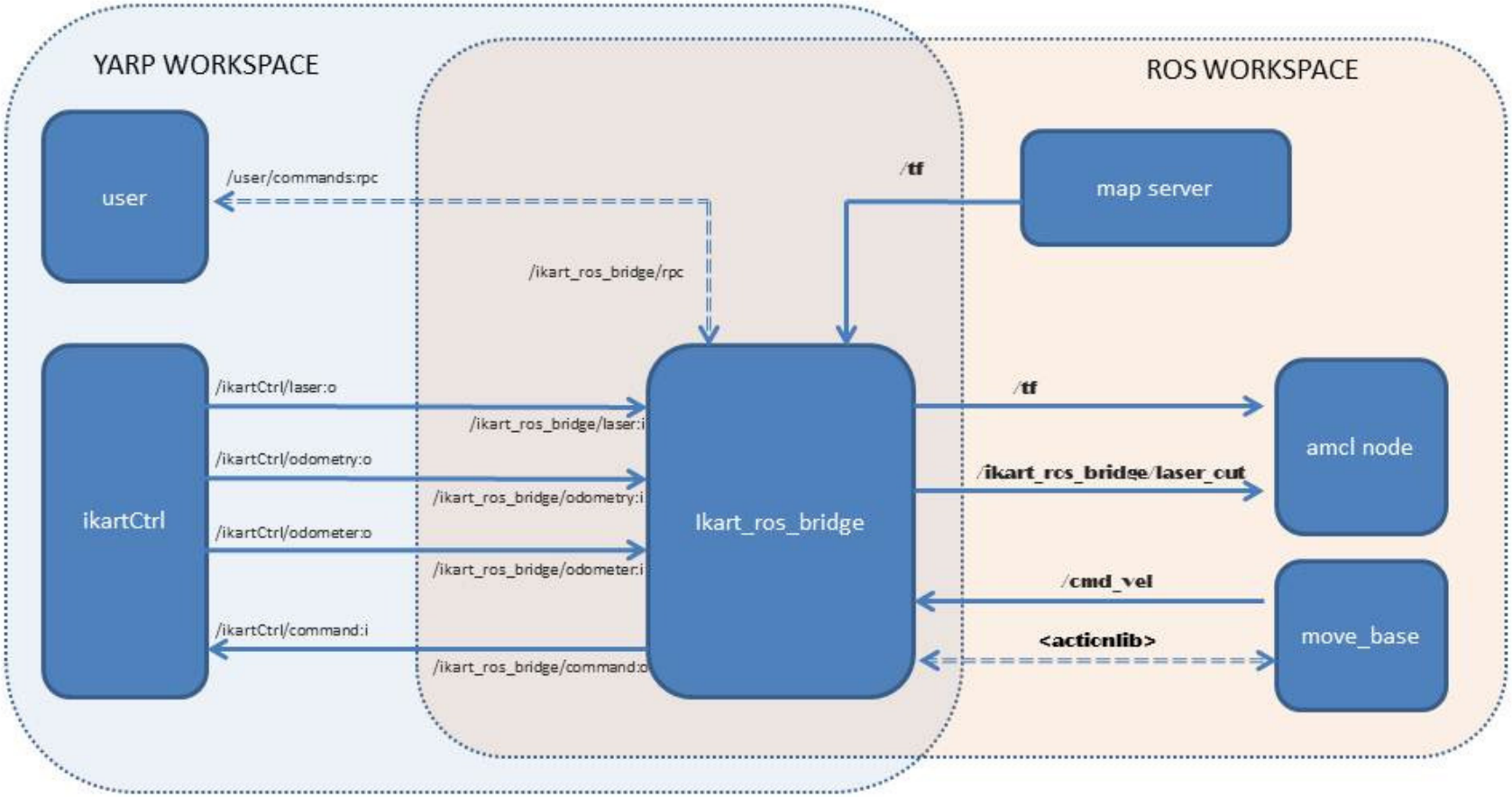
1. Discretely sample in the robot's control space (dx,dy,dtheta)
2. For each sampled velocity, perform forward simulation from the robot's current state to predict what would happen if the sampled velocity were applied for some (short) period of time.
3. Evaluate (score) each trajectory resulting from the forward simulation, using a metric that incorporates characteristics such as: proximity to obstacles, proximity to the goal, proximity to the global path, and speed. Discard illegal trajectories (those that collide with obstacles).
4. Pick the highest-scoring trajectory and send the associated velocity to the mobile base.
5. Rinse and repeat.



iKart

The iKart-ROS bridge

UBUNTU 11.04



iKart

Navigation

Using ROS navigation stack

How to launch it:

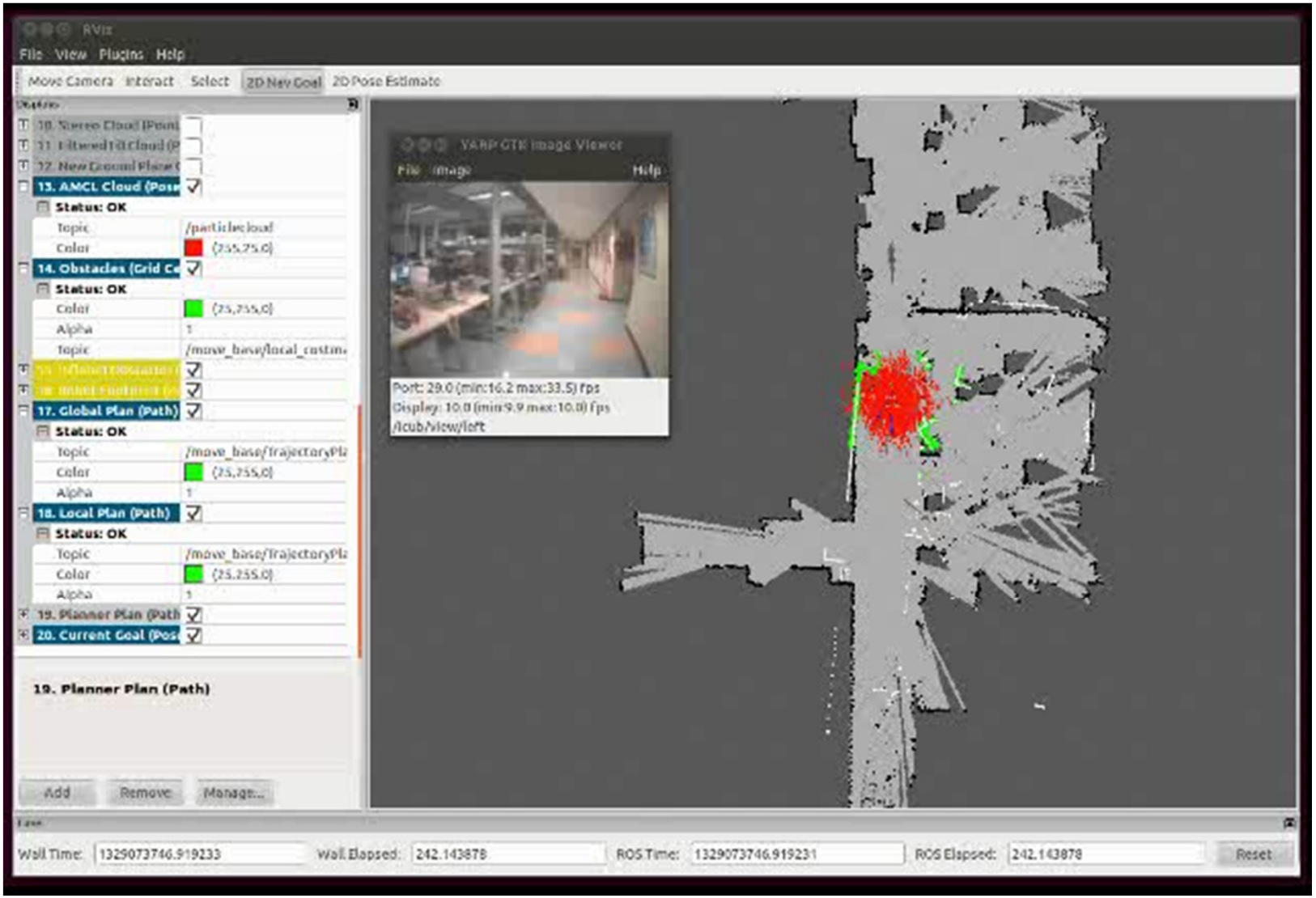
- Verify on the ROS machine that roscore is up and running.
- Verify on the ROS machine that the YARP server running on the iKart can be accessed (yarp detect)
- Start the iKartRosBridge ($IKART_ROS_BRIDGE/bin/iKartRosBridge)
- Start the map server and load the map ($IKART_ROS_BRIDGE/launch/load_maps.sh <map name>)
- Start the localization module (roslaunch $IKART_ROS_BRIDGE/launch/ikart_localize.launch)
- Start the navigation module (roslaunch $IKART_ROS_BRIDGE/launch/ikart_navigate.launch)

How to use it:

- Connect rviz you can send goals using the button 2DNav goal and clicking on the map
- Your application can send commands using the port /ikart_ros_bridge/rpc
- Available commands:
  - help
  - set goal <x> <y> <angle>
  - set frame <name> <x> <y> <angle>
  - stop_navigation
  - get_navigation status
  - …
- Joystick has always the priority!

iKart

Navigation in a known map



iKart

Navigation in a known map

The quality of the obtained trajectories is affected by many parameters, specified in the ROS configuration files.

For localization:

- $IKART_ROS_BRIDGE/launch/ikart_localize.launch

For navigation:

- $IKART_ROS_BRIDGE/config/base_local_planner_params.yaml
- $IKART_ROS_BRIDGE/config/global_costmap_params.yaml
- $IKART_ROS_BRIDGE/config/local_costmap_params.yaml
- $IKART_ROS_BRIDGE/config/costmap_common_params.yaml

The meaning of the parameters is explained in the ROS documentation:

- http://www.ros.org/wiki/gmapping
- http://www.ros.org/wiki/move_base
- http://www.ros.org/wiki/navigation/GlobalPlanner
- http://www.ros.org/wiki/costmap_2d
- http://ros.org/wiki/navfn